

Docket No. AUS990880US1

**METHOD AND SYSTEM FOR PRESENTATION AND MANIPULATION OF
PKCS ENVELOPED-DATA OBJECTS**

CROSS-REFERENCE TO RELATED APPLICATIONS

5

The present invention is related to the following applications entitled "METHOD AND SYSTEM FOR PRESENTATION AND MANIPULATION OF PKCS SIGNED-DATA OBJECTS", U.S.

- Application Serial Number _____, Attorney Docket
10 Number AUS990833US1; "METHOD AND SYSTEM FOR PRESENTATION AND MANIPULATION OF PKCS CERTIFICATION REQUESTS", U.S.
Application Serial Number _____, Attorney Docket
Number AUS990877US1; "METHOD AND SYSTEM FOR PRESENTATION AND MANIPULATION OF PKCS AUTHENTICATED-DATA OBJECTS";
15 U.S. Application Serial Number _____, Attorney Docket
Number AUS990881US1; and all of which are assigned to the same assignee.

BACKGROUND OF THE INVENTION

20

1. Technical Field:

The present invention relates generally to an improved data processing system and, in particular, to a method and apparatus for processing cryptographic data
25 objects formatted according to interoperable standards.

2. Description of Related Art:

Public-key cryptography is the technology in which encryption and decryption involve different keys. The
30 two keys are the public key and the private key, and either can encrypt or decrypt data. A user gives his or

Docket No. AUS990880US1

her public key to other users, keeping the private key to himself or herself. Data encrypted with a public key can be decrypted only with the corresponding private key, and vice versa.

5 As public-key cryptography has gained acceptance, standards have become necessary so that software at two different sites could work together even when the software is developed by different vendors. In particular, standards have been developed to allow
10 agreement on digital signatures, digital enveloping, digital certification, and key agreement. However, interoperability requires strict adherence to communicable formats, and PKCS, or "Public Key Cryptography Standard," provides a basis for
15 interoperable standards in heterogeneous environments.

PKCS is a set of documents published by RSA Laboratories that serves to define data types and algorithms used in public-key cryptography. The first set of ten PKCS standards was released in 1991. In the
20 1993 release PKCS #2 and #4 were incorporated into PKCS #1, so the set of standards included:

PKCS #1: RSA Encryption Standard;
PKCS #3: Diffie-Hellman Key Agreement Standard;
PKCS #5: Password-Based Encryption Standard;
25 PKCS #6: Extended-Certificate Syntax Standard;
PKCS #7: Cryptographic Message Syntax Standard;
PKCS #8: Private-Key Information Syntax Standard;
PKCS #9: Selected Attribute Types; and
PKCS #10: Certification Request Syntax Standard.
30 PKCS continues to evolve and the following standards have been added since 1993:

Docket No. AUS990880US1

PKCS #11: Cryptographic Token Interface Standard;

PKCS #12: Personal Information Exchange Syntax
Standard;

PKCS #13: Elliptic Curve Cryptography Standard; and

5 PKCS #15: Cryptographic Token Information Format
Standard.

Two independent levels of abstraction have been
provided by these standards. The first level is message
syntax, and the second level is specific algorithms. The
10 intention has been that message syntax and specific
algorithms should be orthogonal. In other words, a
standard for the syntax of digitally signed messages
should be able to work with any public-key algorithm, not
just RSA, the public-key algorithm invented by Rivest,
15 Shamir, and Adleman involving exponentiation modulo the
product of two large prime numbers; and a standard for
RSA should be applicable to many different message syntax
standards.

One of these standard documents, PKCS #9, defines a
20 set of attributes that can be used in other PKCS
standards. In particular, PKCS #9 defines selected
attribute types for use in PKCS #6 extended certificates,
PKCS #7 digitally signed messages, PKCS #8 private-key
information, PKCS #12 personal information, and PKCS #15
25 cryptographic token information.

PKCS #7 describes a general syntax for data that may
have cryptography applied to it. In other words, PKCS #7
defines the syntax for several cryptographically
protected messages, including encrypted messages and
30 messages with digital signatures. The syntax admits
recursion, so that one envelope can be nested inside

Docket No. AUS990880US1

another or one party can sign previously enveloped digital data. PKCS #7 also allows arbitrary attributes, such as signing time, to be authenticated along with the content of a message, and it also provides for other
5 attributes, such as countersignatures, to be associated with a signature. A degenerate case of the syntax provides a means for disseminating certificates and certificate-revocation lists. PKCS #7 can also support a variety of architectures for certificate-based key
10 management.

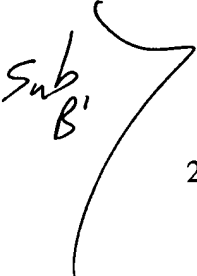
Originally an outgrowth of Internet Privacy-Enhanced Mail, PKCS #7 has become the basis for the widely implemented Secure/Multipurpose Internet Mail Extensions (S/MIME) secure electronic mail specification,
15 an Internet e-mail security standard that employs public key encryption. PKCS #7 has become a basis for message security in systems as diverse as the Secure Electronic Transaction (SET) specification for bank systems.

PKCS #7 is compatible with Privacy-Enhanced Mail
20 (PEM) in that signed-data and signed-and-enveloped-data content, constructed in a PEM-compatible mode, can be converted into PEM messages without any cryptographic operations. A PEM message can similarly be converted into the signed-data and signed-and-enveloped-data
25 content types, i.e. a form that can be processed by applications including or implementations including PKCS #7 without any cryptographic operations. The conversion process is "flat" in the sense that the encapsulated text of the PEM message becomes the "inner" content of the
30 PKCS #7 data. If the encapsulated text happens to contain privacy-enhanced messages, those messages are not

Docket No. AUS990880US1

implemented in the conversion process. PEM can effectively be viewed as a set of encoding rules analogous to the Basic Encoding Rules for ASN.1, abbreviated BER, for PKCS #7 data with these
5 restrictions. Conversion from PKCS #7 to PEM may involve omission of attributes from PKCS #6 extended certificates, which is acceptable since the attributes are not essential to PEM.

The values produced according to PKCS #7 are
10 intended to be BER-encoded. Abstract Syntax Notation One, abbreviated ASN.1, is a notation for describing abstract types and values. The Basic Encoding Rules for ASN.1 give one or more ways to represent any ASN.1 value as an octet string. The Distinguished Encoding Rules for
15 ASN.1, abbreviated DER, are a subset of BER, and give exactly one way to represent any ASN.1 value as an octet string. DER is intended for applications in which a unique octet string encoding is needed, as is the case when a digital signature is computed on an ASN.1 value.
20 ASN.1 and DER encoding are general purpose methods that can be applied to many domains in addition to PKCS.

Sub B 
~~A PKCS #7 EnvelopedData object and the objects that may be contained within a EnvelopedData object are defined in RFC 2630, "Cryptographic Message Syntax," June
25 1999, <http://ietf.org/rfc/rfc2630.txt>. Within this standard, the EnvelopedData definition includes the object version number, the content, a set of certificates, a set of Certificate Revocation Lists (CRLs), and at least one RecipientInfo object that
30 provides per-recipient information.~~

EnvelopedData objects were designed for a

Docket No. AUS990880US1

heterogeneous environment in which the EnvelopedData object and the objects within it can be DER-encoded into a stream of bytes. The DER encoding can be transferred from one system to a completely different system and
5 decoded to reform the EnvelopedData object.

With all the attributes that are part of a EnvelopedData object, administrators, applications developers, and other users can easily be lost in details. They may have access to all the integral
10 objects used in creating a EnvelopedData object, such as the text file to be encrypted, a certificate file, and a private key file, but they may lack the application or means to merge the objects together to create a EnvelopedData object. In other situations, users may
15 receive a EnvelopedData object as an external file or as part of an S/MIME object for which they do not have a targeted application or that they do not wish to be automatically included in a targeted application.

Therefore, it would be advantageous to have an
20 improved method and system for presenting and manipulating secure data objects using interoperable standards within heterogeneous environments, such as using PKCS within a distributed computing environment. It would be still more advantageous to provide users with
25 a method and system to graphically construct a PKCS EnvelopedData object as well as view and manipulate a EnvelopedData object that has been stored or received.

Docket No. AUS990880US1

SUMMARY OF THE INVENTION

A method and system for processing enveloped data objects in a data processing system is presented. The
5 enveloped data object may be formatted, i.e. may maintain a syntax, as defined by PKCS (Public Key Cryptography Standard) standards. An enveloped data object utility allows a user to view and edit the contents of data objects embedded within an enveloped data object via a
10 graphical user interface. Graphical objects represent the data objects embedded within an enveloped data object. A user may drag and drop objects onto other objects within the enveloped data object, and the enveloped data object utility automatically performs the
15 necessary encrypting operations. Logical associations between data objects contained within the enveloped data object, such as between certificates and recipient information objects, are determined or created, and the logical associations are displayed using visual
20 indicators, such arrows or other links, between graphical objects representing the associated data objects. As data objects are added or deleted through user actions on the graphical objects, the visual indicators are updated to reflect any updates to the logical associations
25 between the data objects. The user may direct other operations on the enveloped data object through the graphical user interface.

Docket No. AUS990880US1

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the
5 invention are set forth in the appended claims. The
invention itself, however, as well as a preferred mode of
use, further objectives and advantages thereof, will best
be understood by reference to the following detailed
description of an illustrative embodiment when read in
10 conjunction with the accompanying drawings, wherein:

Figure 1 is a pictorial representation of a
distributed data processing system in which the present
invention may be implemented;

Figure 2A is a block diagram depicting a data
15 processing system that may be implemented as a server;

Figure 2B is a block diagram illustrating the
relationship of software components operating within a
computer system that may implement the present invention;

Figure 3 depicts a dialog window for presenting and
20 manipulating PKCS EnvelopedData objects;

Figures 4A-4D depict a flowchart depicting the
processing of user actions within a dialog window of a
EnvelopedData interface utility for viewing and
manipulating EnvelopedData objects, such as the
25 EnvelopedData object shown in **Figure 3**;

Figure 5A depicts a dialog window for presenting
RecipientInfo object information for a RecipientInfo
object selected by a user within a EnvelopedData interface
utility dialog window;

Figure 5B is the format of a RecipientInfo object
30 data type;

Docket No. AUS990880US1

Figure 6 is a dialog window for presenting attribute information for an attribute selected by a user within a EnvelopedData interface utility dialog window;

Figure 7 depicts a dialog window for presenting
5 certificate information for certificates selected by a user within a EnvelopedData interface utility dialog window; and

Figure 8 depicts a dialog window for presenting
10 certificate revocation list (CRL) information for a CRL selected by a user within a EnvelopedData interface utility dialog window in accordance with a preferred embodiment of the present invention.

US 6,440,000 B2

Docket No. AUS990880US1

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

With reference now to the figures, **Figure 1** depicts a
5 pictorial representation of a distributed data processing
system in which the present invention may be implemented.
Distributed data processing system **100** is a network of
computers in which the present invention may be
implemented. Distributed data processing system **100**
10 contains a network **102**, which is the medium used to
provide communications links between various devices and
computers connected together within distributed data
processing system **100**. Network **102** may include permanent
connections, such as wire or fiber optic cables, or
15 temporary connections made through telephone connections.

In the depicted example, a server **104** and server **106**
is connected to network **102** along with storage unit **108**.
In addition, clients **110**, **112**, and **114** also are connected
to network **102**. These clients **110**, **112**, and **114** may be,
20 for example, personal computers or network computers. For
purposes of this application, a network computer is any
computer, coupled to a network, which receives a program
or other application from another computer coupled to the
network. In the depicted example, server **104** provides
25 data, such as boot files, operating system images, and
applications to clients **110-114**.

Clients **110**, **112**, and **114** are clients to server **104**.
Additionally, clients **110-114** also may be clients to
server **106** in these examples. Distributed data processing
30 system **100** may include additional servers, clients, and

Docket No. AUS990880US1

other devices not shown. In the depicted example, distributed data processing system **100** is the Internet with network **102** representing a worldwide collection of networks and gateways that use the TCP/IP suite of
5 protocols to communicate with one another. At the heart of the Internet is a backbone of high-speed data communication lines between major nodes or host computers, consisting of thousands of commercial, government, educational, and other computer systems that route data
10 and messages. Of course, distributed data processing system **100** also may be implemented as a number of different types of networks, such as, for example, an intranet, a local area network (LAN), or a wide area network (WAN). **Figure 1** is intended as an example of a
15 heterogeneous computing environment and not as an architectural limitation for the present invention.

With reference now to **Figure 2A**, a block diagram depicting a data processing system that may be implemented as a server, such as server **104** or server **106** in **Figure 1**.
20 Data processing system **200** may be a symmetric multiprocessor (SMP) system including a plurality of processors **202** and **204** connected to system bus **206**. Alternatively, a single processor system may be employed. Also connected to system bus **206** is memory
25 controller/cache **208**, which provides an interface to local memory **209**. I/O bus bridge **210** is connected to system bus **206** and provides an interface to I/O bus **212**. Memory controller/cache **208** and I/O bus bridge **210** may be integrated as depicted.
30 Peripheral component interconnect (PCI) bus bridge

Docket No. AUS990880US1

214 connected to I/O bus **212** provides an interface to PCI local bus **216**. A number of modems may be connected to PCI bus **216**. Typical PCI bus implementations will support four PCI expansion slots or add-in connectors.

- 5 Communications links to network computers **108-112** in **Figure 1** may be provided through modem **218** and network adapter **220** connected to PCI local bus **216** through add-in boards.

- Additional PCI bus bridges **222** and **224** provide
10 interfaces for additional PCI buses **226** and **228**, from which additional modems or network adapters may be supported. In this manner, data processing system **200** allows connections to multiple network computers. A memory-mapped graphics adapter **230** and hard disk **232** may
15 also be connected to I/O bus **212** as depicted, either directly or indirectly.

- Those of ordinary skill in the art will appreciate that the hardware depicted in **Figure 2A** may vary. For example, other peripheral devices, such as optical disk
20 drives and the like, also may be used in addition to or in place of the hardware depicted. The depicted example is not meant to imply architectural limitations with respect to the present invention. The data processing system depicted in **Figure 2A** may be, for example, an IBM
25 RISC/System 6000 system, a product of International Business Machines Corporation in Armonk, New York, running the Advanced Interactive Executive (AIX) operating system.

- Alternatively, the operating system may be another commercially available operating system such as JavaOS
30 For Business™ or OS/2™, which are also available from

Docket No. AUS990880US1

IBM. JavaOS is loaded from a server on a network to a network client and supports Java programs and applets. An object oriented programming system such as Java may run in conjunction with the operating system and may
5 provide calls to the operating system from Java programs or applications executing on the data processing system.

The present invention provides a method, a system or apparatus, and computer-implemented instructions for processing cryptographic data objects on a variety of
10 computer platforms and operating systems. Although the present invention could be implemented in most computer languages, it is preferably implemented in Java due to the ability to run Java code in a distributed, heterogeneous environment. Hence, the present invention
15 may operate within a Java runtime environment and operate in conjunction with a Java virtual machine (JVM) yet within the boundaries of a JVM as defined by Java standard specifications. In order to provide a context for the present invention, portions of the operation of a
20 JVM according to Java specifications are herein described.

With reference now to **Figure 2B**, a block diagram illustrates the relationship of software components operating within a computer system that may implement the
25 present invention. Java-based system **250** contains platform specific operating system **252** that provides hardware and system support to software executing on a specific hardware platform. JVM **254** is one software application that may execute in conjunction with the
30 operating system. JVM **254** provides a Java runtime environment with the ability to execute Java application

Docket No. AUS990880US1

or applet **256**, which is a program, servlet, or software component written in the Java programming language. The computer system in which JVM **254** operates may be similar to data processing system **200** described above. However, 5 JVM **254** may be implemented in dedicated hardware on a so-called Java chip, Java-on-silicon, or Java processor with an embedded picoJava core.

At the center of a Java runtime environment is the JVM, which supports all aspects of Java's environment, 10 including its architecture, security features, mobility across networks, and platform independence. The JVM is a virtual computer, i.e. a computer that is specified abstractly. The Java specifications define certain features that every JVM must implement, with some range 15 of design choices that may depend upon the platform on which the JVM is designed to execute. For example, all JVMs must execute Java bytecodes and may use a range of techniques to execute the instructions represented by the bytecodes. A JVM may be implemented completely in 20 software or somewhat in hardware. This flexibility allows different JVMs to be designed for hardware platforms, such as mainframe computers and PDAs.

The JVM is the name of a virtual computer component that actually executes Java programs. Java programs are 25 not run directly by the central processor but instead by the JVM, which is itself a piece of software running on the processor. The JVM allows Java programs to be executed on a different platform as opposed to only the one platform for which the code was compiled. Java 30 programs are compiled for the JVM. In this manner, Java is able to support applications for many types of data

Docket No. AUS990880US1

processing systems, which may contain a variety of central processing units and operating systems architectures.

The data processing systems described above with
5 respect to **Figures 1-2B** may be used to create, modify, transmit, store, and receive cryptographic data objects formatted according to interoperably defined cryptography standards, and in particular, PKCS #7 EnvelopedData objects. The following discussion provides background
10 information on the utility of certificates, signatures, etc.

A certificate is a digital document that vouches for the identity and key ownership of an individual, a computer system, a specific server running on that
15 system, or an organization. For example, a user's certificate verifies that the user owns a particular public key. Certificates are issued by certificate authorities. These authorities are responsible for verifying the identity and key ownership of the
20 individual before issuing the certificate. An identity certificate is a digitally signed statement from one entity, saying that the public key of some other entity has some particular value.

Public keys are numbers associated with a particular
25 entity, and are intended to be known to everyone who needs to have trusted interactions with that entity. An entity is a person, organization, program, computer, business, bank, etc. If some data is digitally signed, it has been stored with the "identity" of an entity and a
30 signature that proves that entity knows about the data. A signature is computed from some data and the private

Docket No. AUS990880US1

key of an entity.

Private keys are numbers that are supposed to be known only to a particular entity, i.e. kept secret. In a typical public key cryptographic system, a private key
5 corresponds to exactly one public key.

Certificates rely on public key cryptographic systems in which (a) private and public keys are paired, (b) private keys are used to sign, and (c) public keys are used to verify signatures. A certificate authority
10 (CA) is an entity (e.g., a business) that is trusted to sign (issue) certificates for other people (entities). It usually has some kind of legal responsibilities for its vouching of the binding between a public key and its owner that allow one to trust the entity that signed a
15 certificate.

There are two basic techniques used to get certificates: (1) make one oneself using the proper software, or (2) ask someone else, such as a certificate authority, to issue one. There are two main inputs to
20 the certificate creation process. The first input is a pair of matched public and private keys generated using some special software. Only the public key is ever shown to anyone else. *One usage of a private key*
~~The private key~~ is used to sign data; if someone improperly knows a private key, they can forge
25 legal documents attributed to a third party. The second input is information about the entity being certified, such as an individual. This normally includes information such as a name and organization address. If a certificate authority issues a certificate, one will
30 normally need to provide proof of identity.

If a certificate authority issues a certificate for

Docket No. AUS990880US1

an individual, the individual must provide a public key and some information about himself. A tool, such as Netscape Navigator, may digitally sign this information and send it to the certificate authority. The

5 certificate authority might be a company that provides trusted third-party certificate authority services. The certificate authority will then generate the certificate and return it. The certificate may contain other information, such as dates during which the certificate
10 is valid and a serial number. One part of the value provided by a certificate authority is to serve as a neutral and trusted introduction service, based in part on their verification requirements, which are openly published in their Certification Service Practices (CSP).

15 The X.509 standard is one of many standards that defines what information can go into a certificate and describes the data format of that information. The "version" field indicates the X.509 version of the certificate format (1, 2, or 3), with provision for
20 future versions of the standard. This identifies which version of the X.509 standard applies to this certificate, which affects what information can be specified in it. Thus far, three versions are defined. Version 1 of the X.509 standard for public key
25 certificates was ratified in 1988. The version 2 standard, ratified in 1993, contained only minor enhancements to the version 1 standard. Version 3, defined in 1996, allows for flexible extensions to certificates in which certificates can be "extended" in a
30 standardized and generic fashion to include additional information. In addition to the traditional fields in

Docket No. AUS990880US1

public key certificates (i.e. those defined in versions 1 and 2 of X.509), version 3 comprises extensions referred to as "standard extensions". The term "standard extensions" refers to the fact that the version 3 X.509 standard defines some broadly applicable extensions to the version 2 certificate. However, certificates are not constrained to only the standard extensions and anyone can register an extension with the appropriate authorities (e.g., ISO). The extension mechanism itself is completely generic.

The "serial number" field specifies the unique, numerical identifier of the certificate in the domain of all public key certificates issued by a particular certificate authority (CA) in order to distinguish one certificate from another. When a certificate is revoked, it is actually the certificate serial number that is posted in a certificate revocation list signed by the certificate authority since posting the entire certificate would be wasteful and completely unnecessary. It is for this reason that the serial number for each certificate in the domain must be unique. The "signature algorithm" field identifies the algorithm used by the certificate authority to sign the certificate. The algorithm identifier, which is a number registered with an internationally-recognized standards organization (e.g., ISO), specifies both the public-key algorithm and the hashing algorithm used by the certificate authority to sign certificates.

The "issuer name" field specifies the X.500 Distinguished Name (DN) of the certificate authority that issued the certificate. For example, the Distinguished

Docket No. AUS990880US1

Name "c=US, o=ACME Corporation" might be used as the Distinguished Name for the certificate authority issuing certificates to the employees of the ACME Corporation in the United States. In some cases, such as root or

5 top-level certificate authority certificates, the issuer signs its own certificates. The "validity period" field specifies the dates and times for the start date and the expiration date of the certificate. Every time a

10 certificate is used, the software should examine the certificate to ensure it is still within its validity period. Each certificate is valid only for a limited amount of time, but this period can be as short as a few seconds or almost as long as a century. The validity period depends on a number of factors, such as the

15 strength of the private key used to sign the certificate or the amount one is willing to pay for a certificate.

The "subject name" field specifies the X.500 Distinguished Name of the entity holding the private key corresponding to the public key identified in the

20 certificate; for example, the Distinguished Name "c=US, o=ACME Corporation, cn=John M. Smith" might be the Distinguished Name for employee John M. Smith of the ACME corporation, where "cn" stands for "common name", "o" is "organization", and "c" is "country".

25 The "public key" field is the public key of the entity being named or identified by the certificate. The "subject public key information" field identifies two important pieces of information: a) the value of the public key owned by the subject, and b) the algorithm

30 identifier specifying the algorithm with which the public key is to be used. The algorithm identifier specifies

Docket No. AUS990880US1

both the public-key algorithm and the hashing algorithm.

The "issuer unique identifier" field was added to the X.509 certificate definition as part of the version 2 standard. The field, which is optional, provides a
5 location to specify a bit string to uniquely identify the issuer X.500 name, in the event that the same X.500 name has been assigned to more than one certificate authority over time.

The "subject unique identifier" field was added to
10 the X.509 certificate definition as part of the version 2 standard. The field, which is optional, provides a location to specify a bit string to uniquely identify the subject X.500 name, in the event that the same X.500 name has been assigned to more than one subject over time
15 (e.g., one John M. Smith leaves ACME Corporation and a second John M. Smith joins ACME Corporation two months later). This field is not used by most certificate authorities for various reasons primarily because there are more convenient ways to uniquely identify a subject.
20 Specifically, most certificate authorities use the serialNumber attribute. Such a scheme fits well within an organization's administrative and directory management procedures because employees require a unique identifier in their X.500 common names anyway (e.g., to handle the
25 case where there are two John M. Smith's in the organization at the same time).

X.509 Version 1 has been available since 1988, is widely deployed, and is the most generic. X.509 Version 2 introduced the concept of subject and issuer unique
30 identifiers to handle the possibility of reuse of subject and/or issuer names over time. Most certificate profile

Docket No. AUS990880US1

documents strongly recommend that names not be reused, and that certificates should not make use of unique identifiers. Version 2 certificates are not widely used.

X.509 Version 3 is the most recent (1996) and
5 supports the notion of extensions, whereby anyone can define an extension and include it in the certificate. Some common extensions in use today are: KeyUsage, which limits the use of the keys for particular purposes such as "signing-only"; and AltNames, which allows other
10 identities to also be associated with this public key, e.g., DNS names, e-mail addresses, IP addresses. Extensions can be marked critical to indicate that the extension should be checked and enforced/used. So, for example, if a certificate has the KeyUsage extension
15 marked critical and set to "keyCertSign" then if this certificate is presented during SSL Communication, it should be rejected, as the certificate extension indicates that the associated private key should only be used for signing certificates and not for SSL.

20 The keys used to interact with various parties need to be hung in a "key chain." In the physical world, a key ring holds keys, and a wallet ^{holds} ~~hold~~ multiple identification and credit cards. In the digital world, a directory service provides storage for digital keys and
25 certificates. The X.500 and LDAP (Lightweight Directory Access Protocol) standards are two main contenders for directory services. Each entry in the directory service is globally and uniquely identified by a Distinguished Name. For example, John M. Smith, who belongs in the
30 Executive Office department at Acme Corporation, might have the following Distinguished Name: "cn=John M. Smith,

Docket No. AUS990880US1

ou=Executive Office, o=ACME Corporation, c=US", where "cn" stands for "common name", "ou" is "organizational unit", "o" is "organization", and "c" is "country".

Second-generation directory services store entries
5 in proprietary file formats, hash, B-tree, or Relational Database Management System. Although RDBMS is not necessarily optimized for X.500 Distinguished Names, the maturity, scalability and additional utilities in RDBMS make it an attractive alternative as a directory service
10 repository. X.509v3 certificates and public keys can also be stored and protected in an X.500- or LDAP-based directory service. If a user's secret key is compromised, the certificate associated with the public key must be revoked and added to the appropriate
15 certificate authority's Certificate Revocation List (CRL).

As noted previously, with all the attributes that are part of a EnvelopedData object, administrators, applications developers, and other users can easily be
20 lost in details. Such users may have access to all the integral objects used in creating a EnvelopedData object, such as the text file to be encrypted, a certificate file, and a private key file, but they may lack the application or means to merge the objects together to
25 create a EnvelopedData object.

The present invention provides a graphical user interface methodology for presenting and manipulating, in particular, PKCS #7 EnvelopedData objects. However, the present invention may be used to view and manipulate
30 cryptographic data objects other than PKCS #7 EnvelopedData objects, assuming that the cryptographic

Docket No. AUS990880US1

data objects are formatted according to interoperably defined cryptography standards with some functional similarity to the PKCS family of standards, such as encapsulated content, certificates, certificate revocation
5 lists, etc. Preferably, the graphical user interface and other functionality described in the following figures may be readily implemented in Java to provide the methodology of the present invention in various interoperable, heterogeneous environments.

10 With reference now to **Figure 3**, a dialog window for presenting and manipulating PKCS EnvelopedData objects is shown in accordance with a preferred embodiment of the present invention. The dialog window provides a visual work area where EnvelopedData objects can be created,
15 changed, etc. The dialog window and associated functions in **Figure 3** and the subsequent figures may be implemented as a stand-alone utility or application, or the dialog window and associated functions may be implemented as an applet within a browser-type application or as a portion
20 of some other type of application.

In the descriptions of the following figures, several common user interface events or actions are mentioned, and it should be noted that equivalent user actions may also be employed, as would be apparent to one of ordinary skill
25 in the art. For example, when a drag and drop operation is mentioned, alternative user actions could be employed, such as selecting the file or data object from a file list box or some other import means. Double-clicking on an object would be equivalent to executing a default action
30 on the object. This action could also be accomplished after right-clicking on the object and selecting the

Docket No. AUS990880US1

default action, which is usually highlighted in a graphically significant way, or via some other means. In general, users can forgo using menus for most operations.

Although areas of the visual display are used to
5 associate similar objects, other manners of visually indicating associated objects may be employed. For example, objects of similar type may have similar colors or shapes that differ from the colors or shapes of other objects of a different type. Additionally, although
10 arrows are shown as visual indicators for relational links between objects, other visual indicators may be employed. For example, similar shape modifications may be made to objects that have a particular relationship. As another alternative, if the objects shown in the
15 dialog are too numerous to clearly show their relationships, the user may optionally remove the relationship indicators, or the user may be required to perform some type of user action to request to see the relationships, e.g. by selecting a button or menu, in
20 which case another window is generated to show relationships. Alternatively, the interface utility could automatically expand the interface utility window in order to obtain the visual area necessary to display relationships between objects.

25 In order to exploit all of the features of the EnvelopedData interface utility shown in **Figure 3**, users typically need to know about certificates and CRLs and, in some cases, the encryption and signing algorithms. However, the interface can pick up most of the values
30 automatically by traversing the objects according to the accepted format or syntax provided the PKCS standards.

Docket No. AUS990880US1

For example, the private key can be matched to its certificate via a token or PKCS #12 file, and encryption and signing algorithms can be stored and retrieved from previous e-mails or user interface actions. It should be
5 noted that files in PKCS #12 format typically store a certificate and its associated private key. The contents of the PKCS #12 file are generally protected by a password or other key mechanism.

Dialog **300** in **Figure 3** is a graphical user interface
10 window for a EnvelopedData interface utility application that allows a user to view and manipulate cryptographic data objects formatted according to interoperably defined cryptography standards, and in particular, PKCS #7 EnvelopedData objects. The user may optionally be
15 provided with functionality to configure the EnvelopedData interface utility with user preferences as to default actions, default methods of displaying certain objects, etc., in a manner known to those skilled in the art of graphical user interfaces.

20 In summary, the enveloped-data content type consists of an encrypted content of any type and encrypted content-encryption keys for one or more recipients. The combination of the encrypted content and one encrypted content-encryption key for a recipient is a "digital
25 envelope" for that recipient. Any type of content can be enveloped for an arbitrary number of recipients using any of the three key management techniques for each recipient.

The typical application of the enveloped-data
30 content type will represent one or more recipients' digital envelopes on content of the data or signed-data

Docket No. AUS990880US1

content types.

Enveloped-data is constructed by the following steps:

1. A content-encryption key for a particular
5 content-encryption algorithm is generated at random.
 2. The content-encryption key is encrypted for each
recipient. The details of this encryption depend on the
key management algorithm used, but three general
techniques are supported:
 - 10 key transport: the content-encryption key is
encrypted in the recipient's public key;
 - key agreement: the recipient's public key and
the sender's private key are used to generate a
pairwise symmetric key, then the content-encryption
15 key is encrypted in the pairwise symmetric key; and
 - symmetric key-encryption keys: the
content-encryption key is encrypted in a previously
distributed symmetric key-encryption key.
 3. For each recipient, the encrypted
20 content-encryption key and other recipient-specific
information are collected into a RecipientInfo value.
 4. The content is encrypted with the
content-encryption key. Content encryption may require
that the content be padded to a multiple of some block
25 size.
 5. The RecipientInfo values for all the recipients
are collected together with the encrypted content to form
an EnvelopedData value.
- A recipient opens the digital envelope by decrypting
30 one of the encrypted content-encryption keys and then
decrypting the encrypted content with the recovered

Docket No. AUS990880US1

content-encryption key.

EnvelopedData **301** provides the name of the EnvelopedData object that is currently being viewed within dialog **300**.

- 5 Version **302** is the syntax version number for the EnvelopedData object being displayed within dialog **300**. As the standards for EnvelopedData objects changes over time, various applications may create EnvelopedData objects in accordance with different versions of the
- 10 standard. The version number is stored within the EnvelopedData object so that an application may know by which version of the standard the EnvelopedData object should be parsed, interpreted, or decoded.

- Attribute list **303** is a list of unauthenticated
- 15 attributes associated with the EnvelopedData object. As there may be more than one attribute, the majority of the attributes may also be hidden within a drop-down list, or the names or identifiers of the attributes may be simultaneously shown within dialog **300**.

- 20 The EnvelopedData interface utility automatically updates the version and attributes as elements of the EnvelopedData object are updated. Elements of an EnvelopedData object are grouped by type into areas of the dialog window so that the user may visually grasp
- 25 associated elements and their relationships. The user can drag and drop an object, such as a text file, onto Content area **350** of the dialog to update the contents with a new value.

- RecipientInfo objects **310-311** are shown grouped
- 30 within an area of dialog **300**. Each RecipientInfo object or recipient information object contains an encrypted key

Docket No. AUS990880US1

that recipients will need to use their private key to unlock. Users can specify their private key by using a PKCS #12 file that contains the private key or by using another token, such as a smart card, that stores the private key. Once the recipients have the unlocked encrypted key, they can decrypt the contents with the encrypted key. The user can select a RecipientInfo object and open a separate visual interface or dialog window to view and manipulate the objects contained within a selected RecipientInfo object, e.g. by double-clicking on a RecipientInfo object.

Certificate objects **321-324** are shown grouped within an area of dialog **300**. This set of certificates are used to help the recipient of the EnvelopedData object to identify the public key that will be needed to verify the contents of the EnvelopedData object. The certificates will typically be in a certificate chain order. Arrows between certificates represent certificate chains that exist within the EnvelopedData object, and arrows **325-326** between the certificate objects show the order from the entry level to the root certificate authority. More than one chain can be present, and not all certificates in the chain need to be present.

Arrows **327-328** between the RecipientInfo objects and the certificates indicate which certificate is related to a particular RecipientInfo object, i.e. the arrows indicate that the public key associated with a certificate was used to encrypt the secret key. For the interface to automatically add a RecipientInfo object, the user would have to add a certificate object to the interface. Along with the previously added or specified

Docket No. AUS990880US1

secret key, the interface would automatically encrypt the secret key with the public key of the added certificate. The interface would create a RecipientInfo object with the encrypted secret key along with other information, such as a reference to the certificate, and place the RecipientInfo object in the interface.

The user can add certificates to the EnvelopedData object by dragging and dropping the certificate object on the visual area. The interface utility will add the certificate to the certificate list. If desired by the user, the interface utility can automatically import the certificates from the configured certificate database to construct the certificate chain for the user after a certificate has been added. The user can select a certificate object and open a separate visual interface or dialog window to view and manipulate the attributes contained within the selected certificate object, e.g., by double-clicking on a certificate.

Certificate Revocation List (CRL) **330** is shown within an area of dialog **300** that is reserved for CRLs. The set of CRLs shown in dialog **300** are used to help the recipient of the EnvelopedData object to identify which certificates should be flagged as invalid. Arrows between a CRL and a certificate object indicate which certificate has been revoked by a particular CRL, and arrow **331** indicates that certificate **322** has been revoked as indicated within CRL **330**. If optionally configured by the user, the interface utility will consult a Lightweight Directory Access Protocol (LDAP) database and use the X.509 names in the certificate and CRL objects to determine which certificates should be revoked. The user

Docket No. AUS990880US1

can add CRLs to the EnvelopedData object by dragging and dropping the CRL object on the CRL visual area. The interface utility will add the CRL to the CRL set. The user can also select a CRL object and open a separate
5 visual interface or dialog window to view and manipulate the attributes contained within the selected CRL object, e.g. by double-clicking on a CRL object.

The encryption key section of dialog **300** contains Generate Encryption Key button **342** that a user can press
10 to automatically generate an encryption key based on the key algorithm specified in encryption key algorithm **340** and parameters **341**. The encryption key algorithm is preferably displayed as a drop-down list that allows a user to choose among several types of encryption key
15 algorithms. Alternatively, the interface could allow the user to select from a simple selection list that offered low, medium, and high values. The interface would take the selection and associate it with predefined algorithm values.

20 Content area **350** displays content **352** that is contained within the EnvelopedData object. Encrypt/Decrypt button **351** is labeled "Encrypt" if the contents of have not yet been encrypted or if the contents were previously decrypted. Encrypt/Decrypt
25 button **351** is labeled "Decrypt" if the contents of have not yet been decrypted or if the contents were previously encrypted. The button can be disabled if a key does not exist, if the contents are empty, or if the encryption key could not be retrieved after an import of the
30 EnvelopedData object because the user did not have the associated private key to decrypt the encrypted secret

Docket No. AUS990880US1

key within a RecipientInfo object.

Existing EnvelopedData objects could be dragged and dropped onto the interface to view a preconstructed EnvelopedData object. The interface could also export a
5 EnvelopedData object that passes validation to a file or other transfer mechanism, such as the clipboard, in a DER-encoded format. Before the EnvelopedData object is exported or stored, the interface will run the defined elements through a set of verification rules, presenting
10 errors to the user if present. The same validation checks will also occur when a EnvelopedData object is imported into the interface. Send button **361** allows the user to send the EnvelopedData object to previously specified e-mail addresses in an S/MIME message. The
15 sender should wrap the message in a Simple Mail Transport Protocol (SMTP). Import button **342** allows the user to import a EnvelopedData object from a specified DER-encoded file and populate the display objects. Export button **343** allows the user to store the EnvelopedData object in a
20 DER-encoded file.

With reference now to **Figures 4A-4D**, a flowchart depicts the processing of user actions within an EnvelopedData interface utility dialog for viewing and manipulating enveloped objects, such as the EnvelopedData
25 object shown in **Figure 3**, in accordance with a preferred embodiment of the present invention. The process begins by waiting for a user action (step **401**). In essence, the flowchart shows a main event loop for processing events within a graphical user interface. The process sits in
30 the event loop monitoring for a user action, processing the user action, and then returning to monitor for

Docket No. AUS990880US1

additional user actions.

A determination is then made as to whether the user has dropped a file object into the Content section of the dialog (step 402). If so, then a determination is made as to whether an encryption key exists (step 403). If so, then the file content is encrypted into an EncryptedContentInfo object and displayed within the Content section (step 404). A Decrypt button is then displayed within the dialog because the Content section now contains an encrypted object (step 405). The process then returns to the main event loop. If an encryption key did not exist, then the file object is displayed within the Content section of the dialog and the Encrypt/Decrypt button is shown as being disabled (step 407). The processing of the file object is then complete, and the process returns to step 401 to await another user action.

Within the main event loop, a determination may be made as to whether a user has requested to delete content, e.g. by selecting an item within the Content section and pressing the delete key (step 408). If so, then the content is removed from the Content section (step 409), and the Encrypt/Decrypt button is shown as being disabled (step 409a). The process then returns to step 401 to await another user action.

Within the main event loop, a determination may be made as to whether a user has requested to add a certificate, e.g. by dragging and dropping a certificate or PKCS #12 file object into the certificate list section (step 410). If so, then a determination is made as to whether a certificate already exists (step 411). If so,

Docket No. AUS990880US1

then no further action is required, and the process returns to the main event loop. If a certificate does not already exist, then the certificate is added to the list of certificates (step 412), and the process checks
5 each certificate in the list to see if the subjectDN for a certificate matches the issuerDN for another certificate (step 413). If not, then the process skips to step 415. If so, then one or more links are drawn between a matching issuerDN certificate and a subjectDN
10 certificate (step 414). A determination is then made as to whether an encryption key exists (step 415). If not, then the process returns to the main event loop. If so, then the user is prompted as to whether a RecipientInfo object should be created (step 416). If not, the process
15 returns to the main event loop. If so, then a RecipientInfo object is created and the encrypted key for the RecipientInfo object is calculated using the public key of the certificate associated with the RecipientInfo object (step 417). The RecipientInfo object is then
20 displayed with an arrow between the RecipientInfo object and the associated certificate (step 418). The processing of the user action on the certificate is then complete, and the process returns to step 401 to await another user action.

25 Within the main event loop, a determination may be made as to whether a user has requested to delete a certificate, e.g. by selecting the certificate and pressing the delete key (step 419). If so, then a determination is made as to whether the certificate is
30 linked to any RecipientInfo objects (step 420). If so, the RecipientInfo object is deleted, and the associated

Docket No. AUS990880US1

link is removed from the display (step **421**). The selected certificate is then deleted from the display (step **422**), and the process returns to the main event loop. If the certificate was not linked to any
5 RecipientInfo objects, then the certificate is simply removed at step **422**, and the process branches back to **401** to await another user action.

Within the main event loop, a determination may be made as to whether the user has attempted to add a
10 certificate replication list (CRL) (step **423**). For example, the user may have dragged and dropped a CRL or PKCS #12 file object into the CRL list section. A determination is then made as to whether the CRL already exists (step **424**). If so, no further action is
15 necessary, and the process returns to the main event loop. If the CRL does not exist, then the CRL is added to the list of CRLs (step **425**). The process then checks each certificate in the certificate list to see if the issuerDN and serialNumber for a certificate matches the
20 issuerDN and serial number in the CRL (step **426**). If not, then the process branches back to the main event loop. If so, then one or more links are drawn between the matching certificates and the CRL (step **427**). The process then returns to step **401** to await another user
25 action.

Within the main event loop, a determination may be made as to whether the user has requested to delete a CRL, e.g. by selecting a CRL and pressing the delete key (step **428**). If so, then a determination is made as to
30 whether the selected CRL is linked to any certificates (step **429**). If so, then the associated link or links are

Docket No. AUS990880US1

removed from the display (step **430**), and the CRL is then deleted (step **431**). The process then returns to the main event loop. If the CRL was not linked to a certificate, then the CRL is simply deleted at step **431**, and the
5 process branches back to step **401** to await another user action.

Within the main event loop, a determination may be made as to whether the user has requested to specify an encryption key, e.g. by dragging and dropping a secret
10 key file object into the Encryption Key section of the dialog (step **432**). If so, then the secret key is decoded and displayed in the Encryption Key section, thereby replacing any existing key (step **433**). A determination is then made as to whether any content exists within the
15 Content section (step **434**). If so, then the content is encrypted with the encryption key and displayed (step **435**). It should be noted that if the contents were previously encrypted with an encryption key, the content would be decrypted with the old encryption key before
20 encrypting with the new key. The Decrypt button is then displayed or enabled because encrypted content now exists (step **436**). After displaying the Decrypt button, or if content did not exist at step **434**, then the process loops through the list of RecipientInfo objects (step **437**).
25 For each RecipientInfo object, the encrypted key for the RecipientInfo object is then recalculated using the public key for the certificate associated with the RecipientInfo object (step **438**). A determination is then made as to whether there are additional RecipientInfo
30 objects to be processed (step **439**). If so, then the

Docket No. AUS990880US1

process branches back to step **437** to process another RecipientInfo object. If not, then the processing of the specified encryption key is complete, and the process branches back to step **401** to await another user action.

- 5 Within the main event loop, a determination may be made as to whether the user has to delete an encryption key, e.g. by selecting an encryption key algorithm from the drop down list on the display and pressing the delete key (step **440**). If so, then the existing encryption key
- 10 is deleted if necessary (step **441**). In response, the Encrypt/Decrypt button is disabled (step **442**). The process then loops through the list of RecipientInfo objects (step **443**) and removes the encrypted key from each RecipientInfo object (step **444**). A determination is
- 15 then made as to whether there are other RecipientInfo objects to be checked (step **445**). If so, then the process branches back to step **443** to process another RecipientInfo object. If not, then the process branches back to step **401** to await another user action.
- 20 Within the main event loop a determination may be made as to whether the user has selected or pressed the Generate Encryption Key button (step **446**). If so, the interface then generates an encryption key based on the encryption key algorithm and displays it within the
- 25 Encryption Key section (step **447**). In response, the Decrypt button is then displayed or enabled (step **448**). The process then loops through the list of RecipientInfo objects (step **449**). For each RecipientInfo object, the encrypted key for the RecipientInfo object is
- 30 recalculated using the public key of the certificate

Docket No. AUS990880US1

associated with the RecipientInfo object (step 450). A determination is then made as to whether there are other RecipientInfo objects to be processed (step 451). If so, the process branches back to step 449 to process another
5 RecipientInfo object. If not, then the process loops back to step 401 to await another user action.

Within the main event loop, a determination may be made as to whether the user has pressed or selected the Encrypt button (step 452). If so, then a determination
10 is made as to whether an encryption key exists (step 453). If not, then no further action is required, and the process returns to the main event loop. If an encryption key exists, then the contents are encrypted with the encryption key and displayed in the Content
15 section (step 454). In response, the Decrypt button is then displayed or enabled (step 455). The process of the user action is then complete, and the process returns to step 401 to await another user action.

Within the main event loop, a determination may be made as to whether a user has selected or pressed the Decrypt button (step 456). If so, then a determination
20 is made as to whether an encryption key exists (step 457). If not, the processing of the user action is complete. If so, then the contents are decrypted with the encryption key and displayed in the Content section
25 (step 458). In response, the Encrypt button is displayed or enabled (step 459). The processing of the user action is then complete, and the process branches back to step 401 to await another user action.

30 Within the main event loop, a determination may be

Docket No. AUS990880US1

made as to whether the user has requested to add an attribute e.g. by right-clicking in the attribute list shown in the dialog (step 460). If so, the interface displays the attribute dialog (step 461). A

- 5 determination is then made as to whether the user has requested to save the attribute (step 462). If not, then the processing is complete; the attribute dialog may be removed, and the process returns to the main event loop. If the user has requested to save the attribute, then it
- 10 is added to the attribute list, replacing any attribute with the same name (step 463). The processing of the user action is then complete, and the process returns to step 401 to await another user action.

- Within the main event loop, a determination may be
- 15 made as to whether a user has requested to view or edit an attribute, e.g. by double-clicking on an attribute on the attribute list (step 464). If so, then the interface displays the attribute dialog with the selected attribute (step 465). At this point, the name of the attribute is
- 20 read-only. The user may edit the value of the attribute as desired. A determination is then made as to whether the user has requested to save the attribute (step 466). If not, then the dialog is removed, and the processing of the user action is complete. If the user has requested
- 25 to save the attribute from the dialog, then the current attribute or modified attribute replaces the selected attribute within the attribute list (step 467). The processing of the user action is then complete, and the process loops back to step 401 to await another user
- 30 action.

Within the main event loop, a determination may be

Docket No. AUS990880US1

made as to whether the user has requested to delete an attribute, e.g. by selecting an attribute from the attribute list and pressing the delete key (step 468). If so, the attribute is removed from the list and the display is updated (step 469). The processing of the user action is then complete, and the process loops back to step 401 to await another user action.

Within the main event loop, a determination may be made as to whether the user has requested to delete a recipient info object, e.g. by selecting a RecipientInfo object and pressing the delete key (step 470). If so, then the RecipientInfo object is deleted and the display is updated (step 471). The processing of the user action is then complete, and the process returns to step 401 to await another user action.

Within the main event loop, a determination may be made as to whether the user has requested to view a certificate, e.g. by double-clicking on a certificate object (step 472). If so, then the interface displays the contents of the certificate in a certificate dialog (step 473). The user then closes the dialog window, and the processing of the user action is complete. The process then branches back to step 401 to await another user action.

Within the main event loop, a determination may be made as to whether the user has requested to view a CRL object, e.g. by double-clicking on a CRL object (step 474). If so, then the interface displays the contents of the CRL in a CRL dialog (step 475). The user may then close the dialog window, and the processing of the user action is complete. The process then branches back to

Docket No. AUS990880US1

step **401** to await another user action.

Within the main event loop, a determination may be made as to whether the user has requested to view a RecipientInfo object, e.g. by double-clicking on a
5 RecipientInfo object (step **476**). If so, then the interface displays the contents of the RecipientInfo object in a RecipientInfo dialog window (step **477**). The user may then close the dialog window, and the processing of the user action is then complete. The process then
10 branches back to step **401** to await another user action.

Within the main event loop, a determination may be made as to whether the user has requested to view the content object, e.g. by double-clicking on the content
15 the contents in the appropriate viewer for the content type of the content object (step **479**). For example, if the content type of the content object is type "text", then the contents may be displayed within a default text editor for the system. The processing of the user action
20 is then complete, and the process then branches back to step **401** to await another user action.

Within the main event loop, a determination may be made as to whether the user has pressed the Send button (step **480**). If so, then a determination is made as to
25 whether there are any RecipientInfo objects (step **481**). If not, then no further action is required, and the process branches back to the main event loop. If RecipientInfo objects exists, then the process loops through the list of RecipientInfo objects (step **482**).
30 For each RecipientInfo object, an S/MIME message containing the EnvelopedData object is sent to the e-mail

Docket No. AUS990880US1

address of the certificate associated with the RecipientInfo object via the SMTP protocol (step 483). A determination is then made as to whether there are other RecipientInfo objects to be processed (step 484). If so, then the process branches back to step 482 to process another RecipientInfo object. If not, then the process branches back to step 401 to await another user action.

Within the main event loop, a determination may be made as to whether the user has pressed the Export button (step 485). If so, then the user is prompted for a file name (step 486), and the EnvelopedData object is then stored in DER-encoded format to the specified file name (step 487). The processing of the user action is then complete, and the process branches back to step 401 to await another user action.

Within the main event loop, a determination may be made as to whether the user has pressed the Import button (step 488). If so, then the user is prompted for a file name (step 489), and the interface imports the EnvelopedData object from the specified DER-encoded file and populates the display objects (step 490). When importing, the interface checks if the user has any private keys associated with the certificates that are linked to the RecipientInfo objects. If so, the interface decrypts the encrypted secret key, and in turn, the secret key is used to decrypt the contents. If not, the interface prompts the user for a private key or PKCS #12 file for a private key that will decrypt the encrypted secret key. If the private key is unsuccessful in decrypting an encrypted secret key for a RecipientInfo or if the user does not specify a private key object, the

Docket No. AUS990880US1

interface will display the EnvelopedData object, but the interface will not show the an encrypted key, the contents will be shown in encrypted format, and the encrypt/decrypt button will be disabled. The processing
5 of the user action is then complete, and the processes branches back to step **401** to await another user action.

Within the main event loop, if no other determination actions have positively identified the user action, then the event is disregarded as an unknown user
10 action (step **491**), and the process branches back to step **401** to continue monitoring for other user actions.

With reference now to **Figure 5A**, a dialog window for presenting RecipientInfo object information is shown for viewing a RecipientInfo object selected by a user within a
15 EnvelopedData interface utility dialog window in accordance with a preferred embodiment of the present invention. Dialog **500** contains Close button **501** for allowing a user to close dialog **500** when the user has finished viewing the RecipientInfo object information.
20 Content value fields **510-515** display the values of various content elements within a RecipientInfo object, which are obtained from the selected certificate by parsing the elements of the RecipientInfo object according to a known, standard format, such as that shown in **Figure 5B**.

25 With reference now to **Figure 5B**, the format of a PKCS RecipientInfo object data type is shown. A RecipientInfo object may be placed into a EnvelopedData object on a per-recipient basis. As is apparent by a simple inspection of both **Figure 5A** and **Figure 5B**, each element
30 within a RecipientInfo object may appear as a content value field within a dialog window for viewing

Docket No. AUS990880US1

RecipientInfo objects, such as dialog **500**. In a similar manner, the certificate and certificate revocation list information may also be shown within a dialog window when these objects are selected by the user, as shown in **Figure 7** and **Figure 8**.

RecipientInfo has a different format for the three key management techniques that are supported: key transport, key agreement, and previously distributed symmetric key-encryption keys. In all cases, the content-encryption key is transferred to one or more recipient in encrypted form. Any of the three key management techniques can be used for each recipient of the same encrypted content. For example, per-recipient information using key transport is represented in the type KeyTransRecipientInfo, as shown in **Figure 5B**. Each instance of KeyTransRecipientInfo transfers the content-encryption key to one recipient.

With reference now to **Figure 6**, a dialog window for presenting attribute information is shown for viewing an attribute selected by a user within a EnvelopedData interface utility dialog window in accordance with a preferred embodiment of the present invention. Dialog **600** contains Close button **601** for allowing a user to close dialog **600** when the user has finished viewing the attribute information. Content value fields **610-612** display the values of various content elements within an attribute, which are obtained from the selected data object according to a known, standard format may vary depending upon the many types of attributes supported in various standards.

With reference now to **Figure 7**, a dialog window for

Docket No. AUS990880US1

presenting certificate information is shown for viewing certificates selected by a user within a EnvelopedData interface utility dialog window in accordance with a preferred embodiment of the present invention. Dialog **700** contains Close button **701** for allowing a user to close dialog **700** when the user has finished viewing the certificate information. Content value fields **710-719** display the values of various content elements within a certificate, which are obtained from the selected certificate by parsing the elements of the certificate according to a known, standard format. Private key symbol **720** shows if the interface has internally associated a private key with the certificate. The private key was derived from the previous EnvelopedData interface. This private key is automatically used by the interface to perform such actions as decrypting encrypted secret keys for EnvelopedData objects. If the interface did not associate a private key with the certificate, the dialog would display a private key icon with a slash symbol through it. The interface will associate the private key with a certificate (the public key in the certificate), if the user dropped a PKCS #12 file on the interface that contained a private key and certificate, for example.

With reference now to **Figure 8**, a dialog window for presenting certificate revocation list (CRL) information is shown for viewing a CRL selected by a user within a EnvelopedData interface utility dialog window in accordance with a preferred embodiment of the present invention. Dialog **800** contains Close button **801** for allowing a user to close dialog **800** when the user has finished viewing the CRL information. Content value

Docket No. AUS990880US1

fields **810-818** display the values of various content elements within a CRL, which are obtained from the selected CRL by parsing the elements of the CRL according to a known, standard format.

5 The advantages of the present invention are apparent in view of the detailed description of the invention provided above. The present invention provides a graphical user interface methodology for presenting and manipulating PKCS #7 EnvelopedData objects. The system
10 automatically decomposes a EnvelopedData object and displays relationships between contained objects for rapid visual comprehension and ease of manipulation by various types of users, including system administrators, network administrators, and application developers. Users have a
15 comprehensive, visual view of the EnvelopedData object and its contents. A user may then easily change and refresh the contents of the EnvelopedData object. Objects can be added via drag and drop operations or through conventional file or socket imports. The present invention is operable
20 in a heterogeneous environment since the methodology can encode and decode DER-encoded objects that may be transmitted to and received from various types of computer systems. As concerns over data security and data integrity become more prevalent with the increasing
25 amounts of e-commerce, the use of cryptographic objects will also grow as they become a desired requirement between merchants that commercially interact.

 It is important to note that while the present invention has been described in the context of a fully
30 functioning data processing system, those of ordinary skill in the art will appreciate that the processes of

Docket No. AUS990880US1

the present invention are capable of being distributed in a form of a computer readable medium of instructions and a variety of forms and that the present invention applies equally regardless of the particular type of signal

5 bearing media actually used to carry out the distribution. Examples of computer readable media include recordable-type media such a floppy disc, a hard disk drive, a RAM, and CD-ROMs and transmission-type media such as digital and analog communications links.

10 The description of the present invention has been presented for purposes of illustration and description, but is not intended to be exhaustive or limited to the invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in
15 the art. The embodiment was chosen and described in order to best explain the principles of the invention the practical application and to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to
20 the particular use contemplated.